DeviceLib

„Der Schlüssel zur intelligenten Gerätevernetzung.“

- Ziel
  - Konzept der DDF Datei
  - Konzept der Gerätebibliothek
- DDF - Device Definition File
  - Allgemein
  - Kommunikationsmethoden
  - Groups und Items
  - Object und IO
  - DDF Grafik
  - Formeln
  - Device im OS
  - Anhang
- DeviceLib Anleitung

HINWEIS

Diese Funktion wird von Slide 1/2 und Base R22 nicht unterstützt.

HINWEIS

Für eine DeviceStation bzw. DDF-Datei wird jeweils ein Device-Lizenzpunkt benötigt.

1 Ziel

Die Device Description Files (DDF) bilden die Grundlage einer strukturierten, einheitlichen Gerätedatenbank. Sie beschreiben Geräteprotokolle, Kommunikationsbefehle, Parameterstrukturen und logische Variablen für eine standardisierte Anbindung externer Geräte.

Das Konzept einer DDF-basierten Gerätedatenbank verfolgt folgende Ziele:

- **Protokollunabhängigkeit:** Ein einheitliches Modell für verschiedene Protokolle wie Modbus, REST, oder ASCII/SERIAL.
- **Wiederverwendbarkeit:** Einmal erstellte DDFs können beliebig vielen Geräten oder Projekten zugeordnet werden.
- **Erweiterbarkeit:** Durch die flexible Struktur lassen sich neue Felder, Protokolle und Funktionen leicht integrieren.
- **Automatische Interpretation:** Systeme können Geräteinformationen, Werte und Befehle direkt aus der DDF-Datei interpretieren, ohne Anpassung des Programmcodes.
- **Wartbarkeit und Transparenz:** Änderungen an der Kommunikation oder Geräteparametern erfolgen ausschließlich durch die Anpassung der DDF-Datei.

Eine vollständige DDF-Datei umfasst typischerweise:

- Allgemeine Geräteeigenschaften (*GENERAL)
- Benutzer Befehle (*COMMAND)
- Datenabfragen und -schreibbefehle (*READ, *WRITE)
- Logische Abbildung der verfügbaren Variablen (*ITEM)
- Dynamische Parameter für URL/Telegrammsteuerung (*ARGS)
- Eingangsverarbeitung von externen REST-Events (*LISTENER)
- Optional Gruppierung von Funktionen (*GROUP)
- Generisches System „Gerät“ (*OBJECT)
- Definition der I/Os beim System „Gerät“ (*IO)
- Individuelle Konfigurations-Eingabefelder (*CONFIG)

Durch diese klare Trennung von Gerätedefinition und Kommunikationslogik wird eine hohe Flexibilität und langfristige Zukunftssicherheit gewährleistet.

1.1 Konzept der DDF Datei

DDF-Datei

- *GENERAL # Gerätedaten
- *READ / *WRITE # Kommunikationsbefehle
- *ITEM # Logische Zuordnung der Variablen
- *ARGS # Dynamische Argumente
- *LISTENER # Empfang externer Ereignisse
- *COMMAND # Befehle in der DeviceStation
- *OBJECT # Generisches System „Gerät“
- *IO # Definition der I/Os
- *CONFIG # Individuelle Konfigurations-Eingabefelder

wird eingelesen

System

- Kommunikation (Modbus, REST, SERIAL,DMX)
- Verarbeitung der Werte nach FORMULA / RFORMULA
- Bereitstellung als Variablen im Steuerungssystem
- Bereitstellung eines Konnektor-Interfaces mittels ALIAS-Namen zur automatischen Kopplung von Funktionsbausteinen
- Event-Auswertung bei REST (Listener)

1.2 Konzept der Gerätebibliothek

Erstellte DDF-Dateien können zentral in die Gerätedatenbank hochgeladen werden. Dort erhalten sie eine eindeutige ID und stehen für Kunden und Tech-niker zum Download bereit.

Workflow zur Erstellung und Bereitstellung einer DDF-Datei

1. **Erstellen einer neuen Gerätebeschreibung inkl. Test mit realem Gerät:**
   Entwicklung und Überprüfung der DDF-Datei im Testaufbau mit dem realen Endgerät.
2. **Anlegen des Geräteprofils in der Gerätedatenbank:**
   Erfassung aller relevanten Gerätedaten (Hersteller, Modell, Protokoll etc.).
3. **Upload der DDF-Datei und eventueller Zusatzinformationen:**
   Hochladen der geprüften DDF-Datei und optionaler ergänzender Dokumente (z.#B. technische Handbücher, Bilder).
4. **Freigeben des Geräteprofils:**
   Überprüfung und endgültige Freigabe des Profils für die allgemeine Nutzung.
5. **Downloaden der DDF und Upload auf das Steuerungssystem:**
   Techniker/Kunden laden die geprüfte DDF herunter und integrieren diese in ihre Systeme.

Dadurch wird sichergestellt, dass alle Systeme auf konsistente, aktuelle und geprüfte Gerätedefinitionen zugreifen können.

Signatur

Die Files werden beim Downloaden aus der DB signiert. Der Schlüssel ist gesichert und kann damit die eindeutige Herkunft nachweisen.

Nur signierte Files können als eigenes Produkt supportet werden.

2 DDF - Device Definition File

Aufbau:

Das DeviceDefinitionFile (DDF) ermöglicht die Protokollbeschreibung des anzubindenden Gerätes. Der strukturelle Aufbau ist für alle Protokollvarianten identisch. Die Inhalte der einzelnen Bereiche können sich jedoch unterscheiden. Der Aufbau ist eine ; getrennte CSV-Tabelle. Da teilweise Zelleninhalte auch ; enthalten müssen solche Zellen mit „ “ eingerahmt werden. Zb. Formeln.

Der Inhalt gliedert sich in:

**GENERAL, *COMMAND, *PREPROCESS(*READ), *ONCHANGE(*WRITE), *LISTENER, *GROUP, *ITEM, *OBJECT, *IO*

In Klammer alternative Namen.

Die Bereiche sind als Tabellen aufgebaut, wobei die Zeile mit dem Bereichsnamen in den Spalten die Tabellen-Spaltennamen enthalten.

Die Bereiche *PREPROCESS(*READ),*ONCHANGE(*WRITE),*LISTENER können auch einen Unterbereich Namens *ARGS beinhalten.

Die Bereiche haben in der 1.Spalte den Bereichsnamen mit *NAME. Bei allen anderen Einträgen muss die 1.Spalte leer. Auch bei Unterbereich *ARGS fängt dieser in der 2.Spalte an.

Der Bereich *GENERAL beinhaltet eine Liste der Variablen/Eigenschaften mit dem Namen in der 2.Spalte und den Wert in der 3.Spalte.

Beispiel:

*GENERAL		
	DEVICE	CAMERA
	MANUFACTURER	GEKKO
	TYPE	Automation Control System

Alle anderen Bereich sind wie folgt aufbebaut. Die Spaltennamen und Inhalte usw. variieren.

*WRITE	ALIAS	METHOD	URL	DATATYPE		
	WRITE	GET	api/v1/var/alarms	JSON		
	*ARGS	METHOD	ALIAS	TYPE		
	ARGS	WRITE	VALUE	arg		
	ARGS	WRITE		arg		
	ARGS	WRITE		arg		
*ITEM	ALIAS	NAME		VISIBILITY	RFORMULA	POLLING
	TEST	VMD			„X.0:=DATA.FIND.VMD.EventNotificationAlert.eventType && (DATA.T > ($.SYS.TIME-2));“	

2.1 Allgemein

GENERAL

In dem Abschnitt sind die verschiedenen Parameter des Gerätes definiert:

- **DEVICE:** Gerätename
- **MANUFACTURER:** Hersteller
- **PROTOCOL:** Protocoll (SERCOM, MODBUS TCP,MODBUS ASCII,MODBUS RTU,REST,DMX512)
- **MODEL_NR:** Modelnummer des Gerätes
- **VERSION_NR:** Versionsnummer des Gerätes

Die Kombination zwischen Protokoll, Model und Version sind je Hersteller eindeutig/einmalig und ergibt die ID des Gerätes.

- **MIN_CONTROL_VERSION:** Mindest Gekko/Controllerversion
- **TIMESTAMP:** Erstellungsdatum

Diese Einträge werden seitens der Datenbank generiert. Die weiteren Parameter hängen vom Protokolltyp ab.

- **DEBUGPORT:** xxx

  xxx= TCP Port. Wird dieser in einem geladenen DDF verwendet, startet im Hintergrund ein Socket-TCP Server auf dem Port.

  Anschließend werden allen DEBUG(xxx); Ausgaben auf diesem Port parallel gesendet.

  Über ein TELENT IP PORT kann dies lokal angezeigt werden.

  Mittels der Eingabe eines Strings mit anschließendem Return wird ein Filter für die Ausgaben gesetzt und nur mehr Zeilen mit diesem Filter werden ausgegeben.

  Eingabe eines leeren Strings/Returntaste löscht den Filter.

  Achtung: Schließen des TELNET-Port deaktiviert den Server, bis zum nächsten Neustart oder laden des passenden DDFs.

HINWEIS

Die Ports für LISTENER_PORT und DEBUGPORT müssen im Bereich 8500–8600 liegen.

Kommunikationsparameter

Allgemein:

- **CONNECTION:** Verbindungstyp
  - SERIALPORT: Zuweisbarer ConrollerPort
  - SERIALPORT+: Zuweisbarer Port ink. AUX und NODE
  - IPADDRESS: Eingebbare IP-Adresse
  - DOMAIN Eingebbare Domain
  - DEFDOMAIN: Feste Domain

Protokoll REST:

- **DOMAIN:** Eingebbare Domainadresse (zb. http:/das/, oder IP je nach Gerät und Verwendung
- **AUTHENTIFICATION:** Authentifizierungstyp
  - NONE
  - PASSWORD

    - DEFPASSWORD: Password im DDF
    - OAUTH_DEVICE_FLOW
    - OAUTH_GRANT_FLOW
- **PASSWORD:** Password bei DEFPASSORD
- **USER:** User bei DEFPASSORD
- **SLAVE:** Standard-Slaveadresse für 1.Device
- **LISTENER_PORT:** TCP-Port für Socketserver
- **SLAVESMAX:** Anzahl maximale Slaves/Devices, welche über UI angegeben werden können. Fehlt der Paramter ist nur 1 Device/Slave.

  Bei SLAVESMAX > 0 wird auf dem UI das Eingabefeld Slaves angezeigt. Dort können mit Komma(,) getrennt die Slaves ID angegeben werden.

  Parallel dazu können bei Domain die einzelnen Devices der Station verschiedenen Domains zugewiesen werden. Die IP-Adressen können auch mit – im letzten Bereich von bis angegeben werden. Zb. 192.168.1-12 legt 12 Geräte an.

Zb. UI-Eingabe:

- DOMAIN: 192.168.1,192.168.1.12-16,192.168.1.19
- SLAVES: CAM1,CAM1,CAM3

Die Slaveadresse wird als Zusatzinfo im Devicenamen angezeigt.

Die maximale Anzahl zwischen Slave-Liste und Domain-Liste bestimmt die Anzahl der Geräte. Begrenzt mit Slavesmax aus dem DDF. Fehlt der Domain, dann wir der vorherige genommen.

Zb. UI-Eingabe:

- DOMAIN: http://192.168.1.2/
- SLAVES: CAM1,CAM1,CAM3

Alle Geräte werden mit derselben Domain angelegt. Diese hat in Kombination mit LISTENER_PORT keinen Sinn. Jedoch wenn ich unterschiedliche REST-Abfragen pro Device desselben Types machen möchte, wie beim RestInterface von GEKKO schon. Zb. Licht1, Licht2,…

Protokoll Modbus:

- **PORT:** Modbus Port. Default = 502
- **SLAVE:** Standard-Slaveadresse für 1.Device
- **SLAVESMAX:** Anzahl maximale Slaves/Devices, welche über UI angegeben werden können.(1..)
- **BAUDRATE:** Baudrate
- **DATABITS:** Databits
- **STOPBITS:** Stopbits
- **PARITY:** NONE,ODD,EVEN
- **FLOWCONTROL:** NO,SW,HW
- **TIMEOUT:** Timeout in ms
- **WAIT_AFTER_CONNECT:** Zeit in sekunden
- **MIN_PAUSE:** Mindestpause in ms
- **WORDORDER:** BIG_ENDIAN,LITTLE_ENDIAN
- **BYTEORDER:** BIG_ENDIAN,LITTLE_ENDIAN
- **ERROR_HANDLING:** FLUSH_OR_RECONNECT_ON_ERROR, NOP_ON_ERROR, FLUSH_OR_CLOSE_ON_ERROR

Protokoll Serial:

- **SLAVESMAX:** Anzahl maximale Slaves/Devices, welche über UI angegeben werden können
- **SLAVE:** Standard-Slaveadresse für 1.Device
- **BAUDRATE:** Baudrate
- **DATABITS:** Databits
- **STOPBITS:** Stopbits
- **PARITY:** NONE,ODD,EVEN
- **FLOWCONTROL:** NO,SW,HW
- **WAIT_AFTER_SEND:** Warte Zeit in sekunden nach senden
- **MIN_PAUSE:** Mindestpause in ms
- **WORDORDER:** BIG_ENDIAN,LITTEL_ENDIAN (nur bei Byte-Telegramm)
- **BYTEORDER:** BIG_ENDIAN,LITTEL_ENDIAN(nur bei Byte-Telegramm)

Beispiel:

*GENERAL		
	DEVICE	Brink Excellent HRV
	MANUFATURER	
	TYPE	Ventilation unit
	PROTOCOL	MODBUS RTU(new)
	MODEL_NR	1
	VERSION_NR	1
	ID	0x0B00004500010000
	MIN_CONTROL_VERSION	
	TIMESTAMP	
	CONNECTION	SERIALPORT+
	PORT	502
	SLAVE	1
	TIMEOUT	5000
	WAIT_AFTER_CONNECT	2
	MIN_PAUSE	50
	WORDORDER	BIG_ENDIAN
	BYTEORDER	LITTLE_ENDIAN
	ERROR_HANDLING	FLUSH_OR_RECONNECT_ON_ERROR
	DEFAULT_ALARMING	

2.2 Kommunikationsmethoden

- **LISTENER**
  Kommunikationsmethoden welche bei Socketverbindungen ausgeführt werden.
- **PREPROCESS (READ)**
  Kommunikationsmethoden welche bei im Intervall ausgeführt werden.
- **ONCHANGE (WRITE)**
  Kommunikationsmethoden welche bei Änderung/Force ausgeführt werden. Ohne Polling intervall.

Der Aufbau ist für alle identisch je nach Protokoll. Spaltennamen unterscheiden sich teilweise.

Protokoll Sercom

- **ALIAS:** Eindeutiger ALIAS für Zugriff auf die Variablen
- **SEND:** Sendetelegramm
- **SEND_DELIMITER_START:** Startkennung als Text oder Hex-Code (/000203)
- **SEND_DELIMITER_END:** Endkennung als Text oder Hex-Code (/000203)
- **RECEIVE_LEN:** Länge Empfangs-/oder Antworttelegramm
- **DELIMITER_START:** Startkennung als Text oder Hex-Code (/000203)
- **DELIMITER_END:** Endkennung als Text oder Hex-Code (/000203)
- **POLLING:** Abfragezyklus in ms

Protokoll Modbus

- **ALIAS:** Eindeutiger ALIAS für Zugriff auf die Variablen
- **FUNCTION:** FC3,FC4,… FC1..16 (Modbusfunktionen)
- **SLAVE:** Slaveadresse (Zahl oder Variable) (optional)
- **ADDRESS:** Startadresse COUNT Anzahl Register
- **POLLING:** Abfragezyklus in ms

Protokoll Rest

- **ALIAS:** Eindeutiger ALIAS für Zugriff auf die Variablen
- **METHOD:** GET/POST/PATCH/LISTEN/PUT/DELETE
- **SSE:** GET/POST/DELETE/PUT/SSE
  Öffnet einen SSE-Stream und empfängt die Events. Aufruf-Parameter idem GET.
- **URL:** Url
  Zusammengesetzte Url für Abfrage= Url + DOMAIN
- **DATATYPE:** JSON/XML
- **FORMULA:** Code, der nach dem Aufruf der Methode ausgeführt wird.
- **POLLING:** Abfragezyklus in ms

FUNCTION Bereich bei MODBUS

*FUNCTION	ALIAS	FORMULA
	READ1	„“

ALIAS und Formel.

Aktuell nur für MODBUS aktive. Damit können in Formula Funktionen definiert werden ähnlich wie bei den normalen Methoden.

Unterschied ist, dass die FUNCTUON mittels Befehl aufgerufen werden und dabei Daten aus dem generellen Block kopiert, bzw. in diesen kopiert werden.

```
FUNCTION(1,’READ1’,GET,'READALL',4,10);
FUNCTION(SLAVE,ZIEL,GET/SET,QUELLE,ADRESSE,ANZAHL);
```

Dieser Aufruf kopiert aus der Modbusblock in READALL ab Register 4 10 Register in den Slave 1 und die Zielfunktion READ1 auf Adresse 0..10.

Damit kann ich einen READ Modbus machen und dann einzelne Blöcke passend abarbeiten.

Oder im SLAVE 1 lese ich alle Register und in den einzelnen Slaves arbeite ich die verschiedenen Blöcke ab, je nach Typ.

SUNSPEC, WAGO,…. Bei SET umgekehrt.

MODBUS Variablen

Man kann mit xxx.ADDRESS und xxx.COUNT die Startregsister und Anzahl setzen und lesen.

Zusatzargumente für REST/SERCOM

- ***ARGS:** Kennzeichner (ARG)
- **METHOD:** ALIAS der zuzuweisenden Funktion , *ALL für Alle
- **ALIAS:** ALIAS des Argumentes: **Ist der Alias leer wird der Namen verwendet.**
- **TYPE:** data, Nur bei REST zusätzlich (url,arg)
- **NAME:**
  - Bei SER: Formatierung zb. %02lldTR1
  - Bei REST: Name des Eintrages.
- **FORMAT:** Formatierung (STRING,DEZIMAL, DOUBLE, T_YYYYMMDDHHMM)
  - DAY: Zeitangabe als Tag. 2025-01-30
  - ISO8601: Zeitangabe auf Basis einer time_t Values
  - T_YYYYMMDDHHMM: Zeitangabe auf Basis einer YYYYMMDDHHMMSS DINT WERTES.
  - STRING: in NAME %s
  - DEZIMAL: in NAME %lld
  - DOUBLE: in Name %d
  - ENUM: val1=CODE1,val2=CODE2,valx=CODEx
    - Wandelt Int in Text. Zb. ENUM:0=STANDARD,2=AT,3=ENGLAND,4=ANYWHERE

  Freie Eingabe: mit „%lld“ oder „%s“ oder „%g“. zb. „S_%lldkW”

  Nur bei REST: Gliederung der Daten bei XML und JSON:
  BLOCK_START / ARRAY_START
  BLOCK_END / ARRAY_END
  Zb.

ARGS	FULL	INHABER	data	INHABER			BLOCK_START
ARGS	FULL	NAME	data	NAME	TEST2		STRING

ARGS	FULL	WERT	data	WERT	12		DEZIMAL
ARGS	FULL	INHABER	data	INHABER			BLOCK_END

Beispiel in JSON:

```
"INHABER": {
  "NAME": "TEST2",
  "WERT": 12
}
```

Beispiel in XML:

```
<INHABER>
    <NAME>TEST2</NAME>
    <WERT>12</WERT>
</INHABER>
```

- **VALUE:** Default-Wert
- **ITEM:** zugewiesenes Item ohne Formel ($..= Zugriff auf GENERALParameter)
  Zb. $SLAVE

Die Reihenfolge der Args bestimmt den Telegrammaufbau (bei SERCOM und REST)
zB.

*PREPROCESS	ALIAS	METHOD	URL	DATATYPE	POLLING			
	FULL	GET	http://	JSON	1000			
	*ARGS	METHOD	ALIAS	TYPE	NAME	VALUE	ITEM	FORMAT
	ARGS	FULL	URL	url	/api/v1/trend/meteo/trend%lld/status	0		DEZIMAL
	ARGS	FULL	TSTART	arg	tstart=%s	2017-02-19T15:00:00+01:00	X.10	T_YYYYMMDDHHMM
	ARGS	FULL	TEND	arg	tend=%s	2017-03-19T15:00:00+01:00	X.11	T_YYYYMMDDHHMM
	ARGS	FULL	DATACOUNT	arg	datacount=%lld	96	X.12	DEZIMAL

2.3 Groups und Items

Groups

Liste der Sub-Devices eines Gerätes. Zb. hat ein Umrichter mehrere Subdevices.

Battery, Netz, PV, House zb. Für den Energiemanager. Aber auch noch Netz-Import, Netz-Export als Zählerdevice oder Batter-Ladung, Entladung.

Die ID wird dann bei den Items diesen als Liste zugewiesen. Dh. ein Item kann auch zu mehreren Gruppen zugewiesen werden. Auch kann der Wert für unterschiedliche Gruppen auch andere Werte haben. Zb. ALIAS=POWER ist Battery_Ladung anders als Netz-Import usw.. Oder Battery-Entladung. Der ALIAS ist dann identisch. Der Name und ID des Items unterschiedlich.

Die Reihenfolge der Funktionen bestimmt die Abarbeitungsfolge.

Beispiel:

*GROUP	ID	ALIAS	NAME
	0	GRID	GRID
	1	BATTERY	BATTERY
	2	PV	PV
	3	HOUSE	HOUSE
	4	GRID_IMPORT	GRID_IMPORT
	5	GRID_EXPORT	GRID_EXPORT
	6	BATTERY_CHARGE	BATTERY_CHARGE
	7	BATTERY_DISCHARGE	BATTERY_DISCHARGE

Items

Liste der Items

- **ALIAS:** Definierter ALIAS für GERÄTE-ZUWEISUNG
- **NAME:** Name welcher im Steuerungssystem erscheint
- **ID:** Eindeutige ID 0..254
- **VISBILITY:** HIDDEN,VISIBLE(Standard=VISIBLE)
- **UNIT:** Einheit
- **GROUP:** siehe Gruppen. List der Gruppen mit , getrennt. zB. 0,1,2,3
- **TYPE:** Leer = STANDARD Wert aus Formula

  ARRAY = Nur bei REST JSON anwendbar. RFORMULA beinhaltet den Namen der direkten Array - Variable.
  Bei Abfrage über das Control wird der Wert des übergebenen Indexes zurückgegeben.

  STRING = Nur bei REST anwendbar. RFORMULA beinhaltet den Namen der direkten Variable.

  RFORMULA: DATA.arrayvar
  getSetDeviceItem(…,DATA.arrayvar,10,…)
  liefert DATA.arrayvar[10] zurück.
- **DEFAULT:** Defaultwert
- **WFORMULA:** Berechungsformel welche beim Schreiben/Ändern der Variable aus dem Steuerungssystem ausgeführt wird
- **RFORMULA:** Berechungsformel welche zyklisch ausgeführt wird
- **POLLING:** Abfrage/Berechnungsintervall in ms

Beispiel:

*ITEM	ALIAS	NAME	VISIBILITY	ID	UNIT	TYPE	DEFAULT	WFORMULA	RFORMULA	POLLING
	DEVICE	Device		0					„X.0:=D02.0.WORD;“	3000
	DEVICE_VER_(B/P)	Device ver. (B/P)		1			12		„X.1:=D04.0.WORD;“	3000
	PRESSURE_EXHAUST_CHNL	Exhaust chnl pressure		5	Pa				„X.5:=D12.0.WORD;“	3000
	FLOW_SETPOINT_[M3/H]	Setpoint flow		6	m^3/h				„X.6:=D13.0.WORD;“	3000
	PRESSURE_IMB_OK	Pressure imbalance ok		7					„X.7:=D16.0.WORD;“	3000
	PRESSURE_FIXED_IMB	Fixed press imbalance		8	m^3/h				„X.8:=D22.0.INT;“	3000
	FLOW_SUPPLY_CUR	Current supply flow		9	m^3/h				„X.9:=D28.0.WORD;“	3000
	FLOW_EXHAUST_CUR	Current exhaust flow		10	m^3/h				„X.10:=D29.0.WORD;“	3000
	BYPASS_FLAP_POS	Bypass flap pos		11					„X.11:=D30.0.WORD;“	3000
	BYPASS_FLAP_FUNC	Bypass flap func		12					„X.12:=D31.0.WORD;“	3000

	PREHEAT_REG_STATUS	Preheat reg status		13					„X.13:=D37.0.WORD;“	3000
	POWER_PREHEAT_REG	Preheat reg power		14					„X.14:=D38.0.WORD;“	3000
	ERROR_CODE_CUR	Current error code		15					„X.15:=D39.0.WORD;“	3000
	FILTER_DISPLAY	Filter display		16					„X.16:=D40.0.WORD;“	3000
	GROUND_HEAT_MODE	Ground heat mode		17					„X.17:=D41.0.WORD;“	3000
	TEMP_GROUND_MIN	Min ground heat temp		18	°C/10				„X.18:=D47.0.INT;“	3000
	TEMP_GROUND_MAX	Max ground heat temp		19	°C/10				„X.19:=D53.0.INT;“	3000
	CO2_SENSOR_NUM	CO2 sensor # (max4)		20					„X.20:=D59.0.WORD;“	3000
	CO2_SENSOR_VALUE	CO2 sensor value		21	ppm				„X.21:=D60.0.WORD;“	3000
	MODBUS_SLAVE_ADDR	Modbus slave addr		30				„W00.0.WORD:=X.30.WORD; W00.F:=1;“		3000

Maskierung

Bei den Items und Groups kann in der Spalte MASK ein String definiert werden.
Dieser wird bei exportieren verglichen mit der Variable je Slave.

`$.ITEMMASK.`

Damit kann man eine Filter definieren mit dem dann bei der IO-Auswahl im OS nur die passenden ITEMS je Slave angezeigt werden, bzw. auch in der Gatewayfunktion und Stationsliste.

$.ITEMMASK == MASK

Type

Die Datentypen Einträge in der Spalte TYPE bei den Items werden für die Gateway-Funktion verwendet zur Datenkonvertierung.

bool
byte
word
dword
lword
usint
uint
udint
ulint
sint
int
dint
lint
real
lreal
string
time

STRING und ARRAY bleiben Ausnahmen für die COMPLEX-ABFRAGE.

2.4 Object und IO

***OBJECT**

Der Reiter *GROUP wird in Kombi mit dem *ITEM für die Plug&Play Device Einbindung benötigt

Der Reiter *GROUP dient parallel dazu im Reiter *OBJECT auch zur Zuordnung des Devices zu Generic-Elementen.

Generic Elemente sind mittels DDF definierte Systemelemente du dienen der grafischen Bedienung und Überwachung von individuellen Geräten.

Bei Generic Element(interner Projektname) handelt es sich um frei definierbare Geräte.

Dh. das DDF definiert zum einen die Kommunikation zum Gerät und mittel *GROUP/*OBJECT die Darstellung als individuelles Gerät im OS.

Die Objekte die angezeigt, bedient und überwacht werden verweisen intern auf die *ITEM-Indexe.

Das Object hat folgende PARAMETER:

*OBJECT				
	GROUP	0…x	Verweis auf die GROUP ID	
	ID		0…19 Fortlaufenden eindeutige ID	
	ALIAS		Angezeigter Aliasnamen	Mehrsprachig vorgesehen mit , getrennte Sprache
	TYPE		1,2,3,4,5,101,103,104	IN: 1=DEZIMAL/ENUM,3=TEXT,4=WERT,5=URL
	ENUM		0…n Anzahl der Enum-Werte	
	ENUMTEXT		Enum-Texte mit , getrennt	zb. TEXT1,TEXT2,TEXT3
	ENUMVAL		Enum-Werte mit , getrennt (Werte != -3)	zb. 0,1,2,3
	MIN		Min WERT	Für Trend und Eingabe
	MAX		Max WERT	Für Trend und Eingabe
	IOTYPE		Aktuell nicht genutzt	
	DIGITS		Nachkommastellen für WERTE Anzeige	
	ITEMID		Verweis auf ITEM-ID	
	UNIT		Einheit	
	ALARM		Alarmvergleich >,>=,<,<=,!=,==	
	ALARMVAL		Alarmwert	
	ALARMTIME		Alarmverzögerung in sec.	
	OUTTYPE		Aktuell nicht genutzt	
	CMDITEMID		Verweis auf Item-ID des Kommandos	

	COMMAND		Anzahl der Kommandos	
	COMMANDENUM		Enum-Texte mit , getrennt	zb. START,STOP,
	COMMANDVAL		Enum-Werte mit , getrennt(Werte != -3)	zb. 0,1,2,3
	VIEWTYPE		Anzeigetyp im UI	0=MAIN,1=STATUS,2=PARAMTER,4=Nur ein Wert als Statusanzeige im Menu des Gerätes
				Nur die Kommandos der Einträge als MAIN werden in den Aktione, Widgets und Uhren als Kommandoauswahl sichtbar

Die Min-/Max-Werte beim Log werden berücksichtigt. Wenn beide Werte 0 sind, wird automatisch der Bereich 0 bis 100 angenommen. Andernfalls werden die minimalen und maximalen Werte aller geloggten Objekte verwendet. Wenn kein Log vorhanden ist, wird der Trend ausgeblendet.

Die Geräte sind aktuell nur als BETAFUNKTION aktivierbar im Menu Einstellungen, Weitere sichtbar!

System „Gerät“ aktivieren:

1. Als Konfigurator anmelden
2. Zahnrad > Einstellungen > Schraubenschlüssel > Weitere > BETA-Funktionen **Ein**

***IO**

Definition von möglichen IO-Zuweisungen im System Gerät

Es können aktuell maximal 10 IO Signale je Gruppe definiert werden. Zugriff in den Formeln erfolgt mit IO.GRP.ID

Die IDs müssen der Reihe nach vergeben werden. 0,1,2,...

Beispiel: „X.1:=IO.1.3;“ Für lesen eines DI,AI bzw. IO.1.2:=1; Für schreiben eines DO,AI

*IO	GROUP	ID	ALIAS	DIR
	1	0	VENTIL1	DO
	1	1	VENTIL2	DO
	1	2	VENTIL	AO
	1	3	SENSOR	DI
	1	4	SENSOR2	AI

2.5 DDF Grafik

Die DDF Dateien erlauben neben der Definition von Algorithmen und Protokollen auch die Definition eines individuellen Unserinterfaces.

Dazu wurde der Bereich Object erweitert. Die Objecteinträge beinhalten nun auch den 3 weitere Spalten: SVG/SVGANIMATE/SVGEVENT.

Bei Zuordung des Devices zu einem Generic-Element(Gerät) werden standardmäßig im Visubereich (0) die ersten 10 Objekte als Listenelemente dargestellt.

Falls beim 1.Object der Gruppe ein SVG/ANIMATE definiert ist, wird dieses Userinterface im Visubereich anstatt und das folgende Element als Listendarstellung dargestellt.

In allen Standard-Elementen kann mittels geladener DDF ein solches Object zugeordnet werden. Damit wird dann die Standard-Darstellung im Visubereich ersetzt.

Wird kein SVGANIMATE definiert/leer wird das über die Spalte SVG definierte SVG dargestellt. Falls ein SVGAnimate definiert ist, muss in diesem falls gewünscht gezielt das SVG mittes Funktion gezeichnet werden.

Falls SVGEVENT definiert ist, werden die Mouseevents an die Formel weitergegeben und ermöglichen damit die Userbedienung zu definieren. In der UserEvent Formel werden die Zeichenfunktionen ignoriert. Das Schreiben von Variablen in dem Bereich setzt ein Repaint für das Element.

Beschreibungen der API siehe DDF-API-Function-TABLE.

DDF Datei-Abschnitt OBJECT:

- *OBJECT
- GROUP ID
- ALIAS
- **SVG**
- **SVGANIMATE**
- **SVGEVENT**

SVG

SVG mit einfacher ‘ innerhalb “”.

```
"<svg id='Livello_1' xmlns='http://www.w3.org/2000/svg' version='1.1' view-
Box='0 0 130 130'>
  <!-- Generator: Adobe Illustrator 29.6.1, SVG Export Plug-In . SVG Version:
 2.1.1 Build 9)  -->
  <defs>
    <style>
      .st0 {
        fill: none;
        stroke: #14387f;
        stroke-miterlimit: 10;
        stroke-width: 3.2px;
      }
    </style>
  </defs>
  <line class='st0' x1='65' y1='14.3' x2='65' y2='107.4'/>
  <line class='st0' x1='19.4' y1='37.5' x2='110.6' y2='37.5'/>
  <line class='st0' x1='19.4' y1='60.8' x2='110.6' y2='60.8'/>
  <line class='st0' x1='19.4' y1='84.1' x2='110.6' y2='84.1'/>
```

```
   <rect  class='st0'  x='3.9'  y='115.1'  width='122.3'  height='8.4'  rx='1'
 ry='1'/>
                                          <path                    class='st0'
 d='M11.6,115.1V7.8c0-.7.6-1.3,1.3-1.3h104.2c.7,0,1.3.6,1.3,1.3v107.3'/>
  <rect class='st0' x='18.4' y='15.2' width='93.1' height='91.2' rx='1' ry='1'
 transform='translate(125.8 -4.2) rotate(90)'/>
</svg>"
```

SVGANIMATE

```
"GUI.setWindow(0, 0, 320, 260, 1.0);
GUI.fillPie(118, 78, 84, 46, 0, 180, 0x33FFF7A1, 0, 0x01000000);
IF $.CTRLVAR.dok > 0 THEN
      GUI.fillPie(120, 78, 80, 50, 0, 180, 0xFFFFE28A, 0, 0xFFFFE28A);
ELSE
     GUI.fillPie(120, 78, 80, 50, 0, 180, 0xFFE0E2E0, 0, 0xFFE0E2E0);
ENDIF;
TEST:=$.CTRLMAINVAR.'alg.lp1.sname';
GUI.text(TEST,160,84,6,'middle','top',0x8F8F0F);
TEST:=$.CTRLROOTVAR.alg.lp1.sname;
GUI.text(TEST,160,104,6,'middle','top',0x8F8F0F);
TEST:=$.CTRLVAR.sname;
GUI.text(TEST,160,124,6,'middle','top',0x8F8F0F);
TEST:=FLOAT_TO_STRING(1.43);
GUI.text(TEST,160,144,6,'middle','top',0x8F8F0F);

IF $.CTRLVAR.iOccLL>0 THEN
    GUI.fillEllipse($.CTRLVAR.iOccLL, $.CTRLVAR.iOccLR, 20, 20, 0x9FFF0000,
 2, 0xFFFF0000);
ENDIF;
"
```

SVGEVENT

```
"IF $.MOUSE.UP==1 THEN
$.CTRLVAR.iOccLL:=$.MOUSE.X;
$.CTRLVAR.iOccLR:=$.MOUSE.Y;
ENDIF;
"
```

2.6 Formeln

Wichtig: Bei Verwendung von ; in den Formeln muss die Formel mit „ “ eigefasst werden. Zudem muss das letzte Zeichen innerhalb der " ein ; sein.

Kommentare

- In den Logiken sind Kommentare frei verwendbar, auch mehrzeilig. Sie müssen in /* */ stehen zb. / * Das ist ein Kommentar */

Zugriff auf Items:

- X.0
  Zugriff auf die Itemvariablen. 0= ID des Items
- Mann kann statt die ITEMS über X.ID anzugeben auch mit den ALIAS arbeiten. ITEM.ALIAS oder wenn Gruppen sind. ITEM.GRPID.ALIAS

Zugriff auf System-Variablen:

- $.SYS
  $.SYS.TIME Systemzeit in Sekunden
  $.SYS.TIME_MS Systemzeit in Millisekunden
- $.PARAM.xxx
  Zugriff auf die Parameter des einzelnen Subdevices.
- $.GPARAM.xxx
  Zugriff auf die Parameter des Hauptdevices.
  z.B. $.GPARAM.MANUFACTURER
- URL: erzeugt einen URL Aufruf. Dieser kann dann dem $.GPARAM.URL übergeben werden und dann als Link in der App aufgerufen werden.
  $.GPARAM.URL:=XXX.URL;

Slaveübergreifende Variablen:

- Mann kann mit $.SLAVE.x.yyy Variablen anderes Slaves setzen und lesen. $.SLAVE.3.$.ITEM-MASK, $.SLAVE.3.READ.ADDRESS , $.SLAVE.3.X.10,…

Zugriff auf Temporäre Variablen:

- Wird der Name der Funktion nicht gefunden wird dies als TEMP Variable verwendet. Variablen gelten nur innerhalb des Devices(Slave)

Zugriff auf Kommunikationsmethoden:

- NNN.xxx
  - NNN= ALIAS der Methode.
  - xxx= Abhängig vom Protokoll

Generelle Variablen:

- F=Forcen/Changed (GET/SET)
- L=Datenlänge
- Q=Qualität
- EC=Errorcounter (GET/SET)
- EN=Error-Nummer
- T=Timesstamp (GET/SET)
- P=Pausieren für x Sekunden (GET/SET)

Modbus:

- xxx.FORMAT
  - xxx=Start-Register
  - FORMAT: Konvertierung WORD/UINT/DWORD/DUINT/LWORD/LUINT/INT/DINT/LINT/FLOAT/DOUBLE

Sercom(Ascii)

- **GET/LESEN**
  - xxx.yyy.FORMAT
    xxx=Start-Byte, yyy=Länge,FORMAT=Format
    FORMAT: INT,FLOAT,BYTESUM(Summe aller Bytes)
  - xxx.zzz
    xxx=Byte
    zzz=String-Enumliste zb. NNN.8.NO,YES,MAYBEE
    Ist der String ab dem 8.Byte gleich mit „NO“ , dann ist Wert 0
    YES=1, MAYBEE=2, Anders =0
- **SET/SCHREIBEN**
  - ARG.xxx
    xxx = ALIAS des Argumentes der Funktion

Rest(JSON)

- **GET/LESEN**
  xxx
  - xxx = VALUE
    Wert des Objektes
    VAL.NNN.NNN.NNN
    NNN= Name des Objektes
  - xxx = ARRAY
    Zugriff auf eine Arrary mit den Folgefunktionen
    ARRAY.MEDIA.NNN.NNN.NNN = Mittelwert
    ARRAY.MAX.NNN.NNN.NNN = Maxwert
    ARRAY.MIN.NNN.NNN.NNN = Minwert
    ARRAY.LEN.NNN.NNN.NNN = Länge des Arrays
    NNN= Name des Objektes
  - xxx = FIND
    Vergleichen des Wertes mit dem folgenden String
    FIND.STRING.NNN.NNN.NNN
    Sucht den STRING im Objektwert
    NNN= Name des Objektes
  - Abfragevariable bei JSON Objekten:
    ARRAY-Wert: mit xxx.xxx[0].sdas[1].das; Mit [].
  - ASLIST: Damit können Werte aus Arrays in eine Liste mit ' getrennt umgewandlet werden.
    zb. XXX.ASLIST.Item[1].subitem[].variable;
    listet die Werte aus dem Array Item[1].subitem der Variable variable auf.
  - xxx.EXIST.yyy.yyy.yyyy
    xxx: DATA
    yyy.yyyy… =JSON Item
    liefert 1 wenn Json/XML Item gefunden wird.
  - xxx.URL
    liefert den URL String des aktuellen Aufrufs
  - xxx.HTTP_CODE
    liefert den http-CODE
  - xxx.HTTP_DATA
    liefert die http-RAW-Daten

- **SET/SCHREIBEN**
  ARG.xxx
  - xxx = ALIAS des Argumentes der Funktion

Aufbau der Formel:

Die Formeln müssen in „ eingeschlossen sein, da sie ; als Befehlszeichen verwenden. Auch newLine ist anwendbar. (tab in Formel “ " erlaubt)

Aufbau:

- Beinhaltet alle Möglichkeiten der aktuellen Formelberechungen. Für die richtige Ausführung ist die konsequente Anwendung von Klammern () wichtig.

Operationen:

*	MUL
/	DIV
+	ADD
-	SUB
^	POT
>	GR
>=	GGL
<	KL
<=	KGL
!=	UGL
==	GL
>>	SHR
<<	SHL
&&	AND
&!	ANDNOT
||	OR
|	BOR
&	BAND
&~	BANDNOT
:=	RESULT

Strukturfunktionen:

- **IF THEN ELSE ENDIF;**

```
"IF X.0 > 100 THEN
    X.1 := X.1 + 1;
ELSE IF X.0 > 10 THEN
    X.1:=X.1+10;
    TESTVAR:=(X.1+X.2)*(10-9);
    DEBUG(TESTVAR,X.2,X.0);
ELSE
    X.1:=X.1+1000;
ENDIF;
```

```
IF X.1 > 1000 THEN
    X.2:=10;
ELSE IF X.1 >200 THEN X.2:=100;  ELSE  X.2:=1000; ENDIF;
X.2:=X.2+10;"
```

- **SWITCH CASE DEFAULT ENDSWITCH;**

```
"DEBUG(X.0);
CASE12 := 10 + 4 - 5;

IF X.0 > 200 THEN
    DEBUG(TEST2);
ELSE IF X.0 > 100 THEN
    DEBUG(TEST2_1);
ELSE
    DEBUG(TEST2_2);

    IF X.0 == 10 THEN
        DEBUG(TEST10);
    ELSE IF X.0 == 11 THEN
        DEBUG(TEST11);
    ELSE
        DEBUG(TEST12);

        SWITCH X.0
            CASE 12:
                DEBUG(CASE12);
            DEFAULT:
                DEBUG(CASE13);
        ENDSWITCH;
    ENDIF;
ENDIF;

DEBUG(TEST3);"
```

- **Forschleife**

```
FOR x:=1 TO y BY 1 DO
    ....
ENDFOR;
```

IF THEN Anweisungen dürfen verschachtelt/kasladiert und es können mehrere IF THEN ELSE IF THEN ELSE IF THEN ELSE ENDIF; angelegt werden.

Funktionsaufrufe in Formeln:

In Formeln ist es möglich Funktionen zu verwenden. Die Funktionen können mehre Parameter haben.

Die gesamte Funktion darf keine Operationszeichen wie +-/…. enthalten

Zb. DEBUG(TEST, TEST); geht nicht, DEBUG(TEST,TEST); ist OK

Aufbau: NNN(XXX,XXX,XXX,XXXX)

- NNN= Name der Funktion
- XXX= Parameter der Funktion. Diese müssen Variablennamen sein und nicht direkt numerische Werte. In dem Fall müssen die Werte zuerst in eine Variable geschrieben werden und dann dieses al Parameter übergeben.

- Wichtig: Die Funktion darf kein Leerzeichen enthalten und auch keine Operationszeichen wie +-/….
- Beispiel:
  X.1:=DATE_DAY($.SYS.TIME);
  Schreibt in X.1 den Monatstag der aktuellen Uhrzeit rein.
  UHRZEIT:=$:SYS.TIME-86000; X.1:=DATE_DAY(UHRZEIT);
  Schreibt in X.1 den Monatstag des Vortages.

Funktionen:

- Bei den Funktionen müssen die Paramter als Variablen übergeben werden. Direkt Werte gehen nicht. (Grund: bei einer Zahl mit punkt kommt das Aufsplitten durcheinander.)
  Beispiel: PARAM_TIME:=10000; X.1:= DATE_MONTH(PARAM_TIME);
- **Datumsfunktionen:**
  PARAM_TIME= VARIABLE mit TIME WERT (Zeit in Sekunden)
  - DATE_MONTH(PARAM_TIME)
  - DATE_YEAR(PARAM_TIME)
  - DATEWDAY(PARAM_TIME)
  - DATE_YDAY(PARAM_TIME)
  - DATE_HOUR(PARAM_TIME)
  - DATE_MIN(PARAM_TIME)
  - DATE_SEC(PARAM_TIME)
- **Zeitfunktionen:**
  - TIME_FROM_YDAY(PARAM_YEAR, PARAM_YEARDAY)
  - TIME_FROM_DATE (YEAR,MON, DAY)
  - TIME_FROM_DATE (YEAR,MON, DAY,HOUR,MIN,SEC);
  - TIMEFROMISO8601(isostring)
- **Math-Funktionen :**
  - LOG(VALUE,BASE)
  - DEC(val) ... GANZZAHL der Zahl
  - DEC(val, div) .... GANZZAHL der Division in Int und real
  - MOD(val, div) ... REST der Division in Int und real.
  - ROUND(val, div) ... GERUNDETER WERT der Division in Int und real
  - ROUND(val) ... GERUNDETER WERT der real zahl

  val und div sind die real-werte
  - SINEFROMTIME(frequency, amplitude, phase=0) frequency in Hz, amplitude, phase in radiant all item oder number
- **Help-Funktionen:**
  - DEBUG(VAR1,VAR2,VAR2,..);
    Die Funktion DEBUG gibt im Log des Device die Namen der Variablen mit dem aktuellen Wert aus.
    Zb. IF .. THEN DEBUG(VAR1,VAR2,X.10); …
    Erscheint im Log: 10:12:30:VAR1=6,VAR2=3,X.10=10.2);
- **String-Funktionen:**
  - X.1:='HALLO';
  - LEN(X.1);
  - ISEQUAL(X.1,X.2);

Andere Operationen wie +-/.. sind nicht möglich. Parameter dürfen hier auch direkte Strings sein zB ISEQUAL(TEST,‘HALLO‘);

- ISEQUAL(string1, string2, [len]) len= Anzahl der zu vergleichenden Zeichen
- SUBSTRING(string, start, [len]) len= Anzahl der Zeichen
- CONCAT(string1, string2,...) string item oder string
- SYSTEMINFO(xxx);
  xxx=NAME Liefert den Hostnamen
  xxx=ID Liefert die Gekko ID
- RANDOMSTRING(xxx);
  xxx= Länge des Strings als Zahl: zb RANDOMSTRING(16);
  Liefert einen Zufallsstring mit 16 Zeichen.
- STRING_TO_NUMBER (string, start, len, type) type=INT8, INT16, INT32, INT64, FLOAT, DOUBLE
- HEXSTRING_TO_NUMBER(STRING,START,LEN,TYPE)
  STRING: VARIABLE oder String’
  START ab 0 des Substrings zum umwandeln
  LEN des Substrings TYPE: INT8(1Byte),INT16(2Byte),INT32(4Byte),INT64(8Byte),FLOAT(4Byte(,DOUBLE(8 Byte)

  Beispiel:

```
TEST:='fl_09818121';
IF ISEQUAL(TEST,'fl_',3) THEN
  VAL:=HEXSTRING_TO_NUMBER(TEST,3,8,FLOAT);
ELSE IF ISEQUAL(TEST,'u6_',3) THEN
  VAL:=HEXSTRING_TO_NUMBER(TEST,3,8,INT64);
ELSE ....
ENDIF;
```

- STRING_TO_NUMBER(STRING,START,LEN,TYPE)
  Idem HEXSTRING_TO..
- SUBSTRING(STRING,START)
- SUBSTRING(STRING,START,LEN)
- REPLACEWITHASCII(Text,Suchtext,ASCIIZeichen);
  TEXT:='HALLO $ANFTEST$ANF';
  TEXT2:=REPLACEWITHASCII(TEXT,'$ANF',34);

- **Speichern:**
  - SAVE_JSON(x,y,c,v,v);
    Speichert Werte in einer Sicherungsdatei, die beim Neustart wieder geladen werden
    Nicht dauernd aufrufen. Achtung.

- DECIMAL_TO_STRING(number)
- FLOAT_TO_STRING(number)
- SETVALUE(prefix, suffix, index, value1, value2,...) prefix as string or variable, suffix part of Variable, index as number or variable, value1..N as number or variable Variable=prefix.index.suffix -> SETVALUE('GET','INT', 10, 30.0); Variable=GET.10.INT;
- SETVALUES(prefix, suffix, index, blocks,value1, …) Blocks : die Werte value1,.... werden x mal geschrieben. zb. bei 2 Werten: entsprechend: prefix.index.suffix,prefix.index+1.suffix,prefix.(1*2)+index.suffix,prefix.(1*2)+index+1.suffix, ...bis blocks
- ISO8601(time,type)
  time in sec
  type=0, ISO mit localtime
  1=ISO mit UTC(hinten Z).

```
TIMESEC:=TIME(0);
ISO:=ISO8601(TIMESEC,0);
DEBUG(ISO);
ISOZ:=ISO8601(TIMESEC,1);
DEBUG(ISOZ);
```

- JSONGETVALUEFROMARRAY(json,arrayname,returnvaluename,LOGIC,TYPE,name,compare,comparevalue[,LOGIC,TYPE,name,compare,comparevalue]);
- PID(INSTANCE,SOLL,IST,P,I,D,N)

2.7 Device im OS

***CONFIG**

Im DDF können maximal 3 Werte als Config angegeben werden, die anschließend im UI (Devcie Station) angezeigt werden. Diese Werte sind retain, d. h. sie werden vom System dauerhaft gespeichert und bleiben auch nach einem Neustart erhalten. Sie dienen der Konfiguration des DDF-Treibers in Ergänzung zu den anderen Parametern.

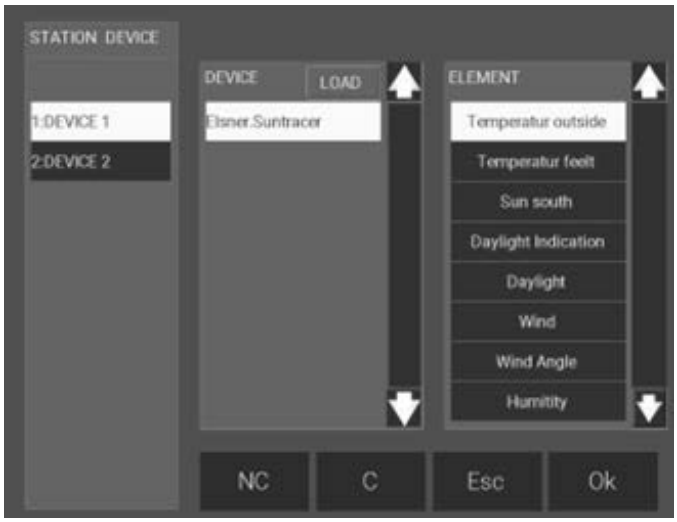
Im Formel-Editor können die Werte über $.CONFIG.ID angesprochen werden, z. B. DEBUG($.CONFIG.0);. Die Beschreibungen werden im UI gepflegt. Direkte Zuweisungen wie $.CONFIG.0 := 1; haben keine Wirkung.

TYPE: 0 = TEXT, 1 = PASSWORD

*CONFIG	ID	ALIAS
	0	TEST
	1	TEST12

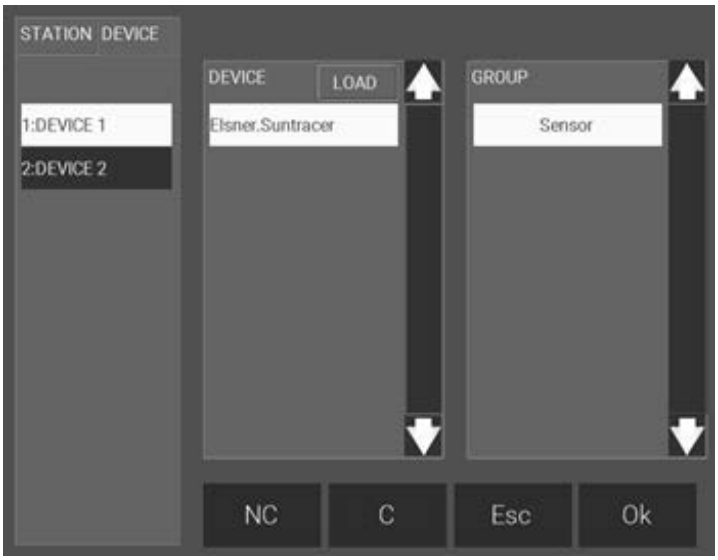
Freie Zuordnung Plug&Play as Device

Die Items können wie jede Ios in allen Systemen/Elementen verwendet werden.



Plug&Play as Device

In verschiedene Elementen können die Devices ausgewählt werden um ein komplettes Gerät oder Subgerät(Gruppe) dem Element zuzuordnen.



Es werden ALIAS verwendet, damit das OS in einem Device nach dem passenden Werten verlinkt.

Abhängig vom Funktionsbaustein sucht das OS im dem Element/Funktion zugeordnetem Device nach den passenden Werten.

z.B. In der Wetterstation sucht das OS automatisch nach TEMPERATUR, SUN_EAST,SUN_WEST,…..

Findet er den Wert wird dieser zugeordnet.

Im Energiezähler sucht das OS den Wert ENERGY und POWER. Die Werte müssen in den passenden Einheiten im Device vorhanden sein.

Im Energiemanager können bis zu vier PV-Zähler und vier Batterien des Typs DeviceLib zugeordnet werden. Bei mehreren Invertern ist eine Zuordnung bei Hauszähler nicht notwenig oder sinnvoll.

Die Kombi aus ALIAS und eventuell Gruppe ergibt die Zuordnung im OS bei den Elementen/Funktionen.

ALIAS NAMEN für Plug&Play as Device

ELEMENT	NAME	UNIT	DIRECTION	NOTE
HEMS_GRID				
	POWER_IMPORT	kW	OUTPUT	
	POWER_EXPORT	kW	OUTPUT	
	ENERGY_IMPORT	Wh	OUTPUT	
	ENERGY_EXPORT	Wh	OUTPUT	
	POWERMAX		OUTPUT	
	QUALITY		OUTPUT	
	ERROR		OUTPUT	
	INFO		OUTPUT	
HEMS_BATTERY				
	POWER_CHARGE	kW	OUTPUT	
	POWER_DISCHARGE	kW	OUTPUT	
	ENERGY_CHARGE	Wh	OUTPUT	
	ENERGY_DISCHARGE	Wh	OUTPUT	
	SOC		OUTPUT	
	POWERMAX		OUTPUT	
	QUALITY		OUTPUT	
	ERROR		OUTPUT	
	INFO		OUTPUT	
HEMS_HOME				
	POWER	kW	OUTPUT	
	ENERGY	Wh	OUTPUT	
	POWER_UNIT		OUTPUT	
	ENERGY_UNIT		OUTPUT	
	POWERMAX		OUTPUT	
	QUALITY		OUTPUT	
	ERROR		OUTPUT	
	INFO		OUTPUT	

ELEMENT	NAME	UNIT	DIRECTION	NOTE
HEMS_PV				
	POWER	kW	OUTPUT	
	ENERGY	Wh	OUTPUT	
	POWER_UNIT		OUTPUT	
	ENERGY_UNIT		OUTPUT	
	POWERMAX		OUTPUT	
	QUALITY		OUTPUT	
	ERROR		OUTPUT	
	INFO		OUTPUT	
HEMS_TARIF				
	REGION		INPUT	STRING
	YEAR		INPUT	ps.2025
	YDAY		INPUT	Day of Year
	GET		INPUT	START QUERY (1=START)
	STATE		OUTPUT	HTTP_CODE (0=WORKING,>0=O-K,<0=ERROR)
	TIMEINTERVALL	sec	OUTPUT	Timeintervall for Array-value(ps. 900=15min Value)
	PRICEFACTOR		OUTPUT	Conversion factor to €/kWh
	PRICELIST		OUTPUT	ARRAY of Pricevalues
	TIMELIST		OUTPUT	ARRAY of Timevalues
	QUALITY		OUTPUT	
	ERROR		OUTPUT	
	INFO		OUTPUT	
ENERGYMETER				
	POWER	kW	OUTPUT	
	ENERGY	Wh	OUTPUT	
	VOLTAGE_L1	V	OUTPUT	

ELEMENT	NAME	UNIT	DIRECTION	NOTE
	VOLTAGE_L2	V	OUTPUT	
	VOLTAGE_L3	V	OUTPUT	
	CURRENT_L1	A	OUTPUT	
	CURRENT_L2	A	OUTPUT	
	CURRENT_L3	A	OUTPUT	
	POWERFACTOR	%	OUTPUT	
	FREQUENCY	Hz	OUTPUT	
	POWER_UNIT		OUTPUT	(Value=kW)
	ENERGY_UNIT		OUTPUT	(Value=Wh)
	QUALITY		OUTPUT	
	ERROR		OUTPUT	
	INFO		OUTPUT	
METER				
	POWER		OUTPUT	(Unit must POWER_UNIT)
	ENERGY		OUTPUT	(Unit must ENERY_UNIT)
	POWER_UNIT		OUTPUT	
	ENERGY_UNIT		OUTPUT	
	QUALITY		OUTPUT	
	ERROR		OUTPUT	
	INFO		OUTPUT	
EMOBIL				
	CURRENT_SET	A	INPUT	
	POWER	kW	OUTPUT	
	ENERGY	Wh	OUTPUT	
	PHASES		OUTPUT	Bitmask (BIT 0=L1,BIT 1=L2,BIT 2=L3)
	CPS		OUTPUT	61851-1 (A,B,C,D,E,F as HexCode)
	VOLTAGE_L1	V	OUTPUT	
	VOLTAGE_L2	V	OUTPUT	

ELEMENT	NAME	UNIT	DIRECTION	NOTE
	VOLTAGE_L3	V	OUTPUT	
	CURRENT_L1	A	OUTPUT	
	CURRENT_L2	A	OUTPUT	
	CURRENT_L3	A	OUTPUT	
	POWER_UNIT		OUTPUT	(Value=kW)
	ENERGY_UNIT		OUTPUT	(Value=Wh)
	QUALITY		OUTPUT	
	ERROR		OUTPUT	
	INFO		OUTPUT	
ACCESS_DOOR				
	UID	ASCII card ID	OUTPUT	
	OPEN		INPUT	
	BLOCKED		INPUT	
	ENABLED		INPUT	
	PERSONID		INPUT	
	AUTHORIZED		INPUT	
	DENIED		INPUT	
	QUALITY		OUTPUT	
	ERROR		OUTPUT	
	INFO		OUTPUT	
WEATHER-STATION				
	WIND	m/s	OUTPUT	
	RAIN		OUTPUT	
	TEMPERATURE	°C	OUTPUT	
	DAYLIGHT	lux	OUTPUT	
	DAYLIGHT_ON		OUTPUT	
	SUN_EAST	klux	OUTPUT	
	SUN_WEST	klux	OUTPUT	
	SUN_SOUTH	klux	OUTPUT	
	QUALITY		OUTPUT	
	TIMESTAMP		OUTPUT	

ELEMENT	NAME	UNIT	DIRECTION	NOTE
	ERRORCODE		OUTPUT	
	ERRORCOUNT		OUTPUT	
	SUMMDATA		OUTPUT	
	CHECKSUM		OUTPUT	
	DATALEN		OUTPUT	
	ERROR		OUTPUT	
VENTILATION/Clima				
	TEMP_AIR_SUPPLY		INPUT	
	TEMP_AIR_EXHAUST		INPUT	
	TEMP_AIR_OUTSIDE		INPUT	
	TEMP_AIR_OUTGOING		INPUT	
	HUMIDITY_AIR_SUPP-LY		INPUT	
	HUMIDITY_AIR_EX-HAUST		INPUT	
	HUMIDITY_AIR_OUTSI-DE		INPUT	
	AIR_QUALITY		INPUT	
	OPERATING_MODE		IN/OUT	
	VENTILATION_POWER		IN/OUT	

2.8 Anhang

Modus-Fehlercode

Fehlercode (Dez)	Bezeichnung
-1	ILLEGAL_FUNCTION
-2	ILLEGAL_DATA_ADDRESS
-3	ILLEGAL_DATA_VALUE
-4	SLAVE_DEVICE_FAILURE
-5	ACKNOWLEDGE
-6	SLAVE_DEVICE_BUSY
-7	NEGATIVE_ACKNOWLEDGE
-8	MEMORY_PARITY_ERROR
-10	GATEWAY_PROBLEM_PATH
-11	GATEWAY_PROBLEM_TARGET
-12	COMM_TIME_OUT
-13	PORT_SOCKET_FAILURE
-14	SELECT_FAILURE
-15	TOO_MANY_DATA
-16	INVALID_CRC
-17	INVALID_EXCPETION_CODE
-18	CONNECTION_CLOSED
-19	TOO_BIG_PACKET
Nicht angegeben	NOT_CONNECTED
	OK

3 DeviceLib Anleitung

Das Portal listet alle im OS integrierten öffentlich verfügbaren Geräte. Jedes Gerät hat eine eindeutige UID welche sich aus dem Hersteller, Model(Gerät), Protokoll und Version der Integration zusammensetzt.

In der Detailansicht der Geräte finden sich die Informationen zum Gerät, eventuelle Herstellerunterlagen und je nach Integrationstyp die Detailinfos. Ziel ist es, dass der Kunde einen Überblick über die Integrationstiefe des Gerätes und der damit verbunden Möglichkeiten zur Verwendung im OS erhält.

Für Geräte die über DDF integriert sind kann der Nutzer die DDF downloaden,um sie dann im OS auf das entsprechende Devices upzuloaden und zu verwenden.
Wird das Gerät eingebunden, passt die die Eingabemaske den Möglichkeiten des Gerätes an und kann darüber parametriert werden.
Das Verhalten ist ähnlich den IO-Stationen. Je Gerät/Device können bis zu 250 Unter/SubDevices angelegt werden. Es können aktuell max.30 Devices angelegt werden.

Configfile

Mann kann über Download/Upload ein Konfigfile laden in welchem Variablen allen Typs definiert sind.

Items, Methoden.xxx, Temp. usw.

Dies muss über GENERAL PARAMETER

CONFIGFILE	ENABLED

Freigeschaltet werden.

Das Configfile wird mitgesichert. Damit lassen sich komplexe Konfigurationen definieren.

DeviceLib Gerät einbinden

1. Portal öffnen und **Gerät suchen**
2. **DDF herunterladen**
3. **Device Station einrichten**:
    - Melden Sie sich am Controller als Konfigurator an und öffnen Sie die Einstellungen.
    - Gehen Sie zum 5. Tab (mit Dreieck-Kreis)
    - Laden Sie die DDF .csv hoch.
    - Eingabefelder ausfüllen und Device-Station aktivieren
4. In den **Systemen** „DEVICE-DB“ Gerät auswählen und das gesamte Gerät, eine Gruppe oder auch nur einzelne Datenpunkte zuweisen.